WEEK -4 UNDERSTANDING DOCUMENT

Adds new functionality, enhances existing features, interacts with third-party apps, automates processes.

    - **Scripts run on:**
        - Client-side: Runs in the browser, affects UI (e.g., field messages, making fields read-only or mandatory).
        - Server-side: Runs in the backend, triggered by database actions, ACL processing, script includes, etc.
        - MID Server: Executes scripts in certain integrations.
- **Performance Consideration:**
    - Scripting affects system performance; use only if necessary.
    - Avoid scripting if 80% of the requirement can be met with low-code/no-code tools.
- **ServiceNow Syntax Editor:**
    - Built-in editor with contextual help, syntax coloring, auto-completion, and debugging.
    - Enabled by default.
- **Programming Language:**
    - JavaScript is used for both client and server-side scripting.

Client-Side Scripting:

- **Purpose:** Modify form UI; runs in the browser.
- **Impact:** Can slow form load times; use sparingly.
- **Types:**
    - onLoad: Runs when the form loads, pre-populates fields.
    - onChange: Triggers when a user modifies a field.
    - onSubmit: Validates form before submission.
    - onCellEdit: Watches list view fields for changes.
- **Script Execution Order**: Determined by the "Order" field; lower values run first.
- **Client APIs:**
    - **GlideForm (g_form):** Manages form fields and methods.
    - **GlideUser (g_user):** Accesses session user details.
    - **Scratchpad (g_scratchpad):** Temporarily holds data between display rules and scripts.
- **g_form Methods:**
    - getValue(), setValue(), showFieldMsg(), addInfoMessage(), clearValue(), isNewRecord().

- **Client Debugging:**
    - Tools include alert(), try/catch, response time indicators.

UI Policies:

- Control form field behavior (hide, mandatory, read-only).
- Can be configured through condition builder or advanced scripting.
- UI policies can also leverage g_form, g_user, and g_scratchpad.

Server-Side Scripting:

- **Business Rules (BR)**: Run on the server when the database is manipulated or queried.
    - **Triggers**:
        - insert → Runs on record insertion.
        - update → Runs on record updates.
        - delete → Runs on record deletion.
        - query → Runs when querying the database.
- **BR Advanced View**:
    - **When**: Specifies when the business rule executes.
    - **Order**: Determines the execution order if multiple rules exist for a table.
- **Business Rule Objects**:
    - current, previous, g_scratchpad.
- **Execution Types**:
    - **Before**: Runs synchronously before querying the database, preventing certain records from displaying.
    - **Display**: Fetches server-side data into g_scratchpad for use by client scripts.
    - **After**: Runs after the database is queried.
    - **Async**: Executes asynchronously without blocking other user operations.
- **Debugging Tools**: try/catch, script debugger, tracer, console debugging, GlideSystem methods.

GlideSystem (gs):

- **User Methods**:
    - getUser(), getUserID(), hasRole(), hasRoleInGroup().
- **System Methods**:
    - getProperty(), getReference(), log(), print(), debug(), eventQueue().
- **Date and Time Methods**:

- beginningOfLastWeek(), nowDateTime(), minutesAgo(), etc.

GlideRecord:

- Used to query data from the database.
- **Syntax**:
  - var records = new GlideRecord('<table_name>');
  - my_obj.addQuery('active', '=', 'true');
  - my_obj.query();
  - while(my_obj.next()){} (for iterating through records).
  - my_obj.update() (for updating records).
- **Additional Queries**:
  - Use addOrCondition() for OR conditions.
  - For a single record: my_obj.get(<condition>).
- **GlideAggregate()**: Handles aggregate functions like count().
- **addEncodedQuery()**: Uses encoded filters from condition builder.
- **GlideQuery**: 100% JavaScript, fail-fast, expressive.

Script Includes (SI):

- **Purpose**: Reusable code that remains dormant until invoked.
- **Types**:
  - **Single Function**: Not callable from the client-side.
  - **Class (Collection of Functions)**: Callable from the client-side.
  - **Extended Class**: Extendable and callable.
- **Client Callable Classes**:
  - Extend from AbstractAjaxProcessor to receive server data.
  - Use glideAjax to call server-side code from client scripts or UI policies.
- **glideAjax**:
  - Add parameters via addParam().
  - Retrieves XML response using getXML() or getXMLAnswer().
  - For JSON results: json.stringify() on the server side, and json.parse(response) on the client-side.

Script Includes Script

```
var HelloWorld = Class.create();
HelloWorld.prototype = Object.extendsObject(AbstractAjaxProcessor, {
alertGreeting: function() {
return "Hello " + this.getParameter('sysparm_user_name') + "!";
 }
});
```

Client-side Script

```
var greeting = new GlideAjax('HelloWorld');
greeting.addParam('sysparm_name','alertGreeting');
greeting.addParam('sysparm_user_name',"Ruth");
greeting.getXML(HelloWorldParse);

function HelloWorldParse(response) {
var answerFromXML = response.responseXML.documentElement.getAttribute("answer");
alert(answerFromXML);
}
```